

计算机学院学科基础课

计算机组成

机器语言 (3)

高小鹏

北京航空航天大学计算机学院
系统结构研究所

提纲

- 内容主要取材：CS61C的7讲
 - <http://inst.eecs.berkeley.edu/~cs61c/su12>
- 存储型程序概念
- R-指令格式
- I-指令格式
 - 分支和PC相关寻址
- J-指令格式

Great Idea #1: Levels of Representation/Interpretation

Higher-Level Language
Program (e.g. C)

Compiler

Assembly Language
Program (e.g. MIPS)

Assembler

Machine Language
Program (MIPS)

*Machine
Interpretation*

Hardware Architecture Description
(e.g. block diagrams)

*Architecture
Implementation*

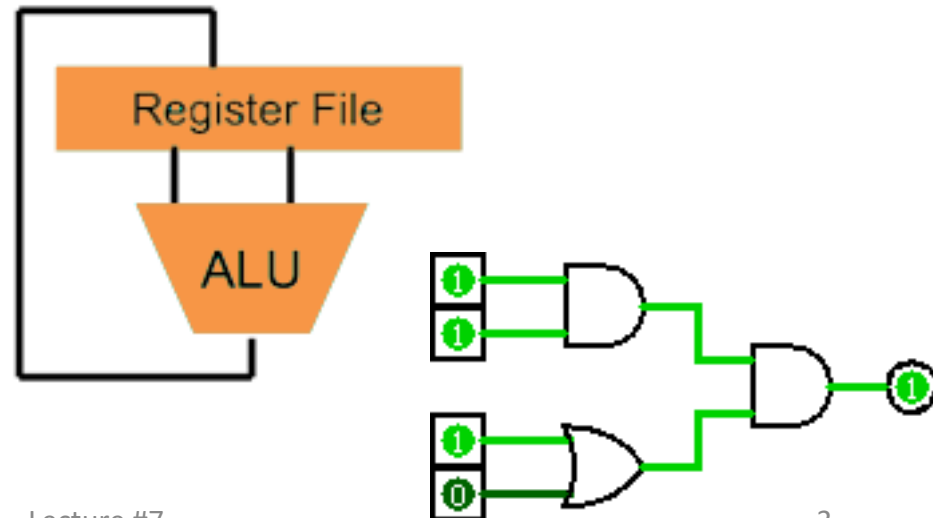
Logic Circuit Description
(Circuit Schematic Diagrams)

```
temp = v[k];  
v[k] = v[k+1];  
v[k+1] = temp;
```

```
lw    $t0, 0($2)  
lw    $t1, 4($2)  
sw    $t1, 0($2)  
sw    $t0, 4($2)
```

We
are
here

```
0000 1001 1100 0110 1010 1111 0101 1000  
1010 1111 0101 1000 0000 1001 1100 0110  
1100 0110 1010 1111 0101 1000 0000 1001  
0101 1000 0000 1001 1100 0110 1010 1111
```



MIPS-C 指令集

(版本: 1.0)

文档编号: COCO01

高小鹏

北京航空航天大学计算机学院

2014 年

4. ADDU: 无符号加

	31	26	25	21	20	16	15	11	10	6	5	0
编码	special 000000	rs	rt	rd	0 00000	addu 100001						
	6	5	5	5	5	6						
格式	addu rd, rs, rt											
描述	$GPR[rd] \leftarrow GPR[rs] + GPR[rt]$											
操作	$GPR[rd] \leftarrow GPR[rs] + GPR[rt]$											
示例	addu \$s1, \$s2, \$s3											
其他												

5. AND: 与

	31	26	25	21	20	16	15	11	10	6	5	0
编码	special 000000		rs		rt		rd		0 00000		and 100100	
	6		5		5		5		5		6	
格式	and rd, rs, rt											
描述	$GPR[rd] \leftarrow GPR[rs] \text{ AND } GPR[rt]$											
操作	$GPR[rd] \leftarrow GPR[rs] \text{ AND } GPR[rt]$											
示例	and \$s1, \$s2, \$s3											
其他												

6. ANDI: 与立即数

	31	26	25	21	20	16	15	0	
编码	andi 001100		rs		rt		immediate		
	6		5		5		16		
格式	andi rt, rs, immediate								
描述	$GPR[rt] \leftarrow GPR[rs] \text{ AND } \text{immediate}$								
操作	$GPR[rt] \leftarrow GPR[rs] \text{ AND } \text{zero_extend}(\text{immediate})$								
示例	andi \$s1, \$s2, 0x55AA								
其他									

提纲

- 内容主要取材：CS61C的7讲
 - <http://inst.eecs.berkeley.edu/~cs61c/su12>
- 存储型程序概念
- R-指令格式
- I-指令格式
 - 分支和PC相关寻址
- J-指令格式

程序存储概念

- ❑ 指令以二进制方式被编码
- ❑ 程序存储在存储器中；可以从存储器中读取程序也可以写入程序
 - ◆ 存储方式与数据存储完全相同
- ❑ 简化了计算机系统的软件/硬件设计
- ❑ 存储器技术既可以存储数据，也可以存储程序
- ❑ 由于存储在存储器单元中，因此指令和数据都有地址



二进制兼容

- ❑ 程序是以二进制形式发布的
 - ◆ 指令集与程序之间是强相关
- ❑ **新机器**不仅能运行基于**新指令**编译产生的**新程序**，同样也最好能运行**老程序**
- ❑ 上述特性被称为向后兼容（backward compatible）
 - ◆ 示例：今天的i7处理器仍然能运行1981年在8086处理器上编译产生的程序



把指令当做数看待^{1/2}

- ❑ 假设：所有的数据都是以字为单位的(32位)
 - ◆ 每个寄存器是字宽度的
 - ◆ `lw`和`sw`读写主存的单位是字
- ❑ 问题：如何用二进制表示指令？
 - ◆ 计算机只能理解0和1，无法理解“`add $t0, $0, $0`”
- ❑ 回答：数据以字为单位，每条指令同样被编码为一个字！
 - ◆ MIPS的所有指令的二进制编码宽度均为32位

把指令当做数看待^{2/2}

T
设计越规则
实现越简单

- 指令的32位被划分为若干域
 - ◆ 域：占据若干特定位；代表特定含义
 - ◆ 同一个域在不同指令的含义大体是相同的
- 指令的32位解读与数据的32位数是不同的
 - ◆ 数据的32位是作为一个整体被解读
 - ◆ 指令的各个域分别表示指令的不同信息

MIPS的3类指令格式

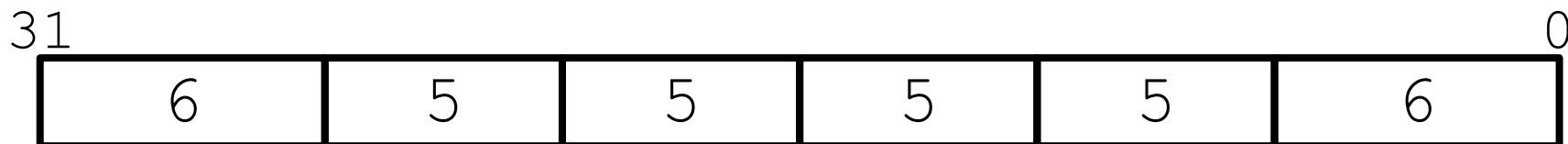
- **I型指令**：指令中包含立即数
 - ◆ lw/sw的偏移是立即数；beq/bne同样包含有偏移
 - ◆ srl等移位指令：也有5位立即数（移位位数），但不属于I型指令
- **J型指令**：j和jal
 - ◆ jr：不是J型指令
- **R型指令**：所有其他的指令

提纲

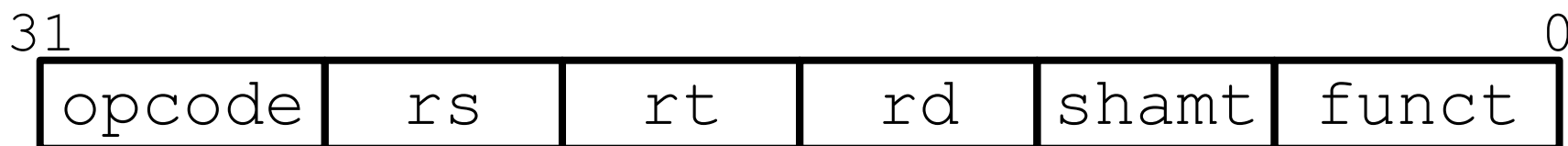
- 内容主要取材：CS61C的7讲
 - <http://inst.eecs.berkeley.edu/~cs61c/su12>
- 存储型程序概念
- R-指令格式
- I-指令格式
- 分支和PC相关寻址
- J-指令格式

R型指令^{1/3}

- 指令包含6个域： $6 + 5 + 5 + 5 + 5 + 6 = 32$



- 为便于理解，每个域都有一个名字



- 每个域都被视为无符号整数

- ◆ 5位域表示范围为：0~31
- ◆ 6位域表示范围为：0~63

R型指令^{2/3}

- `opcode`(6): 代表指令操作
 - ◆ R型指令的opcode固定为0b000000
- `funct`(6): 与opcode组合, 精确定义指令的具体操作
 - ◆ 主要是服务于R型指令
- Q: MIPS最多可以有多少条R型指令?
 - ◆ 由于opcode固定为0, 因此funct的编码数决定了最大条数: 64
- Q: 为什么不将opcode和funct合并为一个12位的域呢?
 - ◆ 后续内容将回答这个问题



R型指令^{3/3}

- **rs** (5): 指定1st操作数 (source寄存器)
- **rt** (5): 指定2nd操作数 (target寄存器)
- **rd** (5): 指定结果回写的寄存器 (destination寄存器)
- MIPS的寄存器个数是32, 因此5位无符号数就可以表示
 - ◆ 这种编码方式非常直观: `add dst, src1, src2` → `add rd, rs, rt`
 - ◆ 注意: 与具体指令相关, 有的域是无效的
- **shamt** (5): 移位指令中的移位位数
 - ◆ 由于寄存器只有32位, 因此移位位数大于31没有意义
 - 移位位数大于31, 结果必然为0。既然如此, 直接赋值为0而无需移位
 - ◆ 注意: 除了移位指令, 该域固定为0

R型指令示例^{1/2}

❑ MIPS指令

```
add    $8, $9, $10
```

❑ 伪代码

```
add    R[rd] = R[rs] + R[rt]
```

❑ 构造各域

opcode = 0 (查手册)

funct = 32 (查手册)

rd = 8 (目的寄存器)

rs = 9 (1st寄存器)

rt = 10 (2nd寄存器)

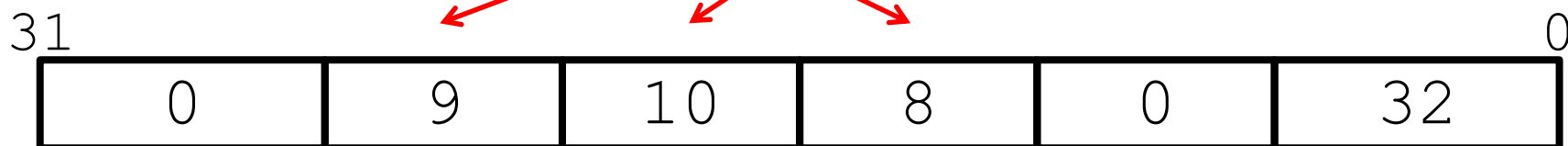
shamt = 0 (不是移位指令)

R型指令示例^{2/2}

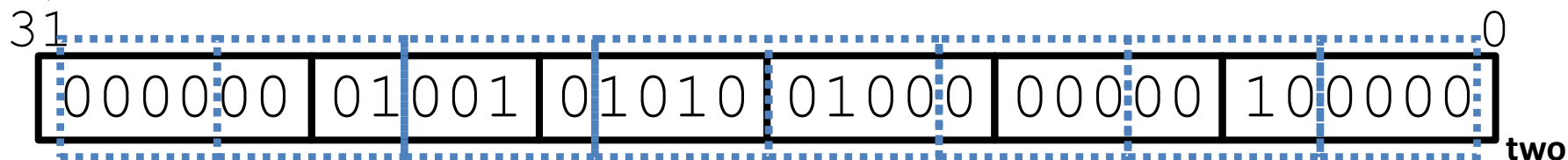
指令:

add \$8, \$9, \$10

域 (10进制)



域 (2进制)



16进制: 0x 012A 4020

10进制: 19,546,144

二进制编码: 机器码

通常不这么
表示指令

NOP

- 0x00000000是什么指令?
 - ◆ opcode: 0, 所以这是一条R型指令
- 根据指令手册, 机器码对应的指令是
sll \$0, \$0, 0
 - ◆ Q: 指令是什么功能?
- NOP是一条特殊指令
 - ◆ NOP主要用于解决流水线冲突

提纲

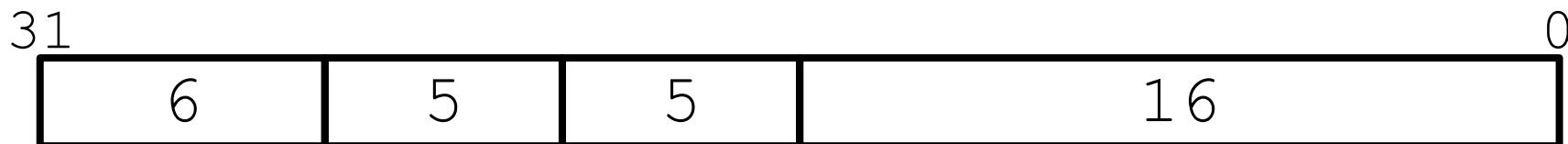
- 内容主要取材：CS61C的7讲
 - <http://inst.eecs.berkeley.edu/~cs61c/su12>
- 存储型程序概念
- R-指令格式
- I-指令格式
 - 分支和PC相关寻址
- J-指令格式

I型指令^{1/4}

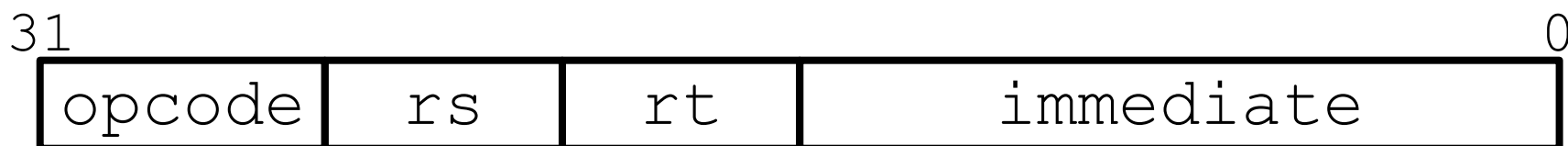
- 指令的立即数指的是什么？
 - ◆ 5位或6位的数字，太短，不被认为是立即数
- 理想的，MIPS最好只有一种指令格式
 - ◆ 很遗憾，我们必须在指令格式上进行折中
- 但是，在定义新的指令时，应该尽可能使得新指令格式与R型指令格式尽可能保持一致
 - ◆ 不难想象，如果使用了立即数，那么指令能用的寄存器最多只有2个

I型指令^{2/4}

- 指令包含6个域： $6 + 5 + 5 + 16 = 32$



- 域的命名



- 前3个域与R型指令相同
 - 最重要的是opcode域保持在相同的位置

I型指令^{3/4}

□ **opcode**(6): 代表指令操作

◆ I型指令的opcode为非零编码: 总共可以编码64 (2^6) 条指令



◆ R型指令用opcode和funct两个独立域有助于保持格式的兼容性

□ **rs** (5): 指定1st操作数 (source寄存器)

□ **rt** (5): 指定2nd操作数 (target寄存器)



◆ target并非都是“目的”。例如: sw的rt就是“读”

□ **immediate** (16): 无符号or有符号

◆ 无符号: 位运算指令 (如and/or/nor等)、小于置位指令 (如slti等)

• `zero_ext()`: 运算前需要进行无符号扩展

◆ 有符号: 分支指令 (如beq/bne等)、访存指令 (如lw/sw等)

• 以word为单位

• `sign_ext()`: 运算前需要进行符号扩展

Q

对于lw/sw, 16位立即数是否够用?

I型指令示例^{1/2}

❑ MIPS指令

```
addi    $21, $22, -50
```

❑ 伪代码

```
addi    R[rt] = R[rs] + sign_ext(immediate)
```

❑ 构造各域

opcode = 8 (查手册)

rs = 22 (1st寄存器)

rt = 21 (2nd寄存器)

immediate = -50 (10进制或16进制表示均可)

T

```
addi $rt, $rs, immediate
```

I型指令示例^{2/2}

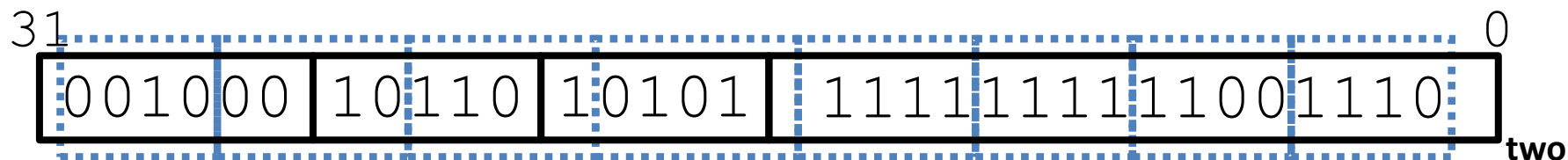
指令:

addi \$21, \$22, -50

域 (10进制)



域 (2进制)



16进制: 0x 22D5 FFCE

T

addi \$rt, \$rs, immediate

如何计算32立即数？

- 32位立即数的应用场景，如：
 - ◆ `addi/slti/andi/ori`：等需要计算2个32位数
 - ◆ `lw/sw`：在使用前，需要先初始化基地址寄存器
- 解决方案：不改变指令格式，而是增加一条新指令
- **Load Upper Immediate** (`lui`)
 - ◆ `lui reg, imm`
 - ◆ `reg`的高16位写入`imm`，低16位写入0
 - RTL: $R[\text{reg}] \leftarrow \text{imm} \parallel 0^{16}$



lui 示例

- 需求: `addi $t0, $t0, 0xABABCD`



这是一条伪指令!

- 会被assembler转换为3条指令

```
lui  $at, 0xABAB      # 高16位
ori  $at, $at, 0xCDCD  # 低16位
add  $t0, $t0, $at     # 赋值
```

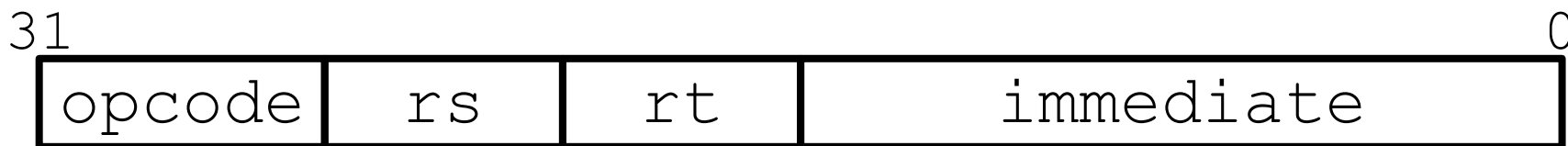
T
只应由assembler
使用\$at

- 通过增加lui, MIPS可以用16位立即数来处理任意大小的数据



分支指令（B类指令）

□ beq和bne



- ◆ 比较rs和rt，并根据比较结果决定是否转移



- 并非所有的B类指令的rt域都是寄存器！

- ◆ 需要指定要转移的地址


□ 分支指令主要用于构造：if-else，while，for

- ◆ 转移的范围通常很小（< 50指令）
- ◆ 函数调用和无条件跳转用J型指令

□ Q：如何用immediate表示地址？

- ◆ 由于指令存储在主存中，而主存单元可以由基地址+偏移的方式定位
- ◆ B类指令的基地址就是PC（即当前这条B类指令的PC值）

PC相对寻址

- **PC相对寻址**：PC为基地址，**immediate**为偏移（二进制补码）
 - ◆ 基本计算方法： $PC \leftarrow PC + \text{偏移}$
 - ◆ 关键是：偏移值如何得到
- 下一条指令的PC值的计算方法
 - ◆ 比较结果为**真**： $PC = PC + 4$
 - ◆ 比较结果为**假**： $PC = (PC + 4) + (\text{immediate} * 4)$
- Q1：为什么**immediate**乘以4？
 - ◆ 存储器是以字节为单位的
 - ◆ 指令都是32位长，且指令是字对齐，这意味着最低2位恒为0
 - ◆  **immediate**没有必要记录最低2位，乘以4后就得到了对应字节地址！
- Q2：为什么基地址是PC+4，而不是PC？
 - ◆ 这与硬件设计有关，后续内容讲解

immediate表示的转移范围是否足够大？

- **immediate**为16位符号数，其表示范围是正负 $\pm 2^{15}$
- 由于转移范围为 $\pm 2^{15}$ 字，意味着转移的指令数为 $\pm 2^{15}$ 条
 - ◆ **immediate**省略了最低2位，即量纲为字
- 按照1行C代码对应10条指令，则可转移的C代码块大小为3000行！

B类指令示例^{1/2}

❑ MIPS指令

```
1  Loop: beq    $9, $0, End
2      addu   $8, $8, $10
3      addiu  $9, $9, -1
4      j      Loop
5      End:
```

从beq后面的那条指令 (addu) 计算



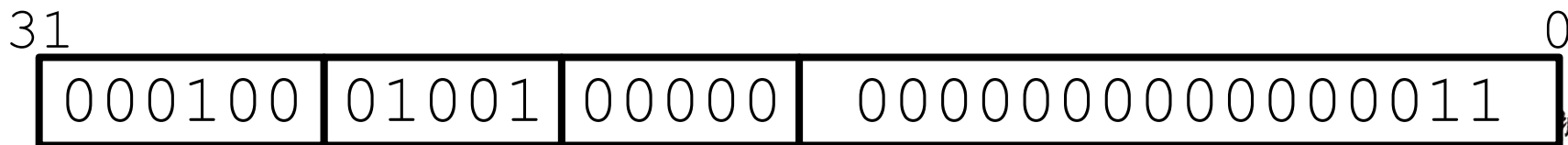
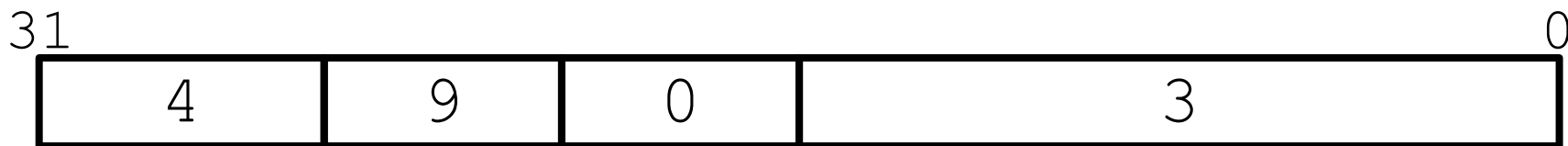
❑ 构造各域

opcode = 4 (查手册)

rs = 9 (1st寄存器)

rt = 0 (2nd寄存器)

immediate = 3



更深入的思考

- 如果移动代码，是否会导致B类指令的**immediate**域的变化？
 - ◆ 会变化：如果移动某行而非B指令所涉及的整块代码
 - ◆ 无变化：如果移动B指令所涉及的整块代码
- 如果跳转的目的地址超出了 2^{15} 条指令，该怎么办？
 - ◆ 组合B类指令与J类指令

```
beq $s0, $0, far
```

```
# next instr
```

-->

```
bne $s0, $0, next
```

```
j far
```

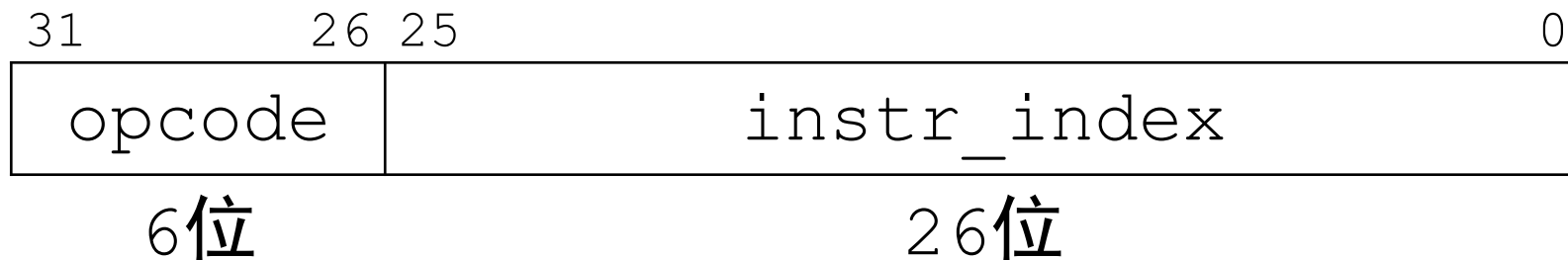
```
next: # next instr
```

提纲

- 内容主要取材：CS61C的7讲
 - <http://inst.eecs.berkeley.edu/~cs61c/su12>
- 存储型程序概念
- R-指令格式
- I-指令格式
 - 分支和PC相关寻址
- J-指令格式
- 汇编实战
- 反汇编实战

J型指令

- J指令只定义了2个域



- 要点

- ◆ opcode的位置及位数与R型、I型指令保持一致
- ◆ 其他域合并在一起构造大的地址范围

J型指令

$PC \leftarrow PC_{31..28} || instr_index || 0^2$

❑ Q: J指令的转移范围有多大?

❑ 分析

- ◆ PC的高4位来自当前指令的高4位
- ◆ 意味着目的地址与当前指令必在同一区段{X000_0000, XFFF_FFFF}
- ◆ 故j指令的转移范围是256MB

❑ 由此可知, 程序员无法用j指令跳转到256MB以外的地址空间

❑ 只有jr可以跳转到4GB内任意地址



◆ jr: R型指令

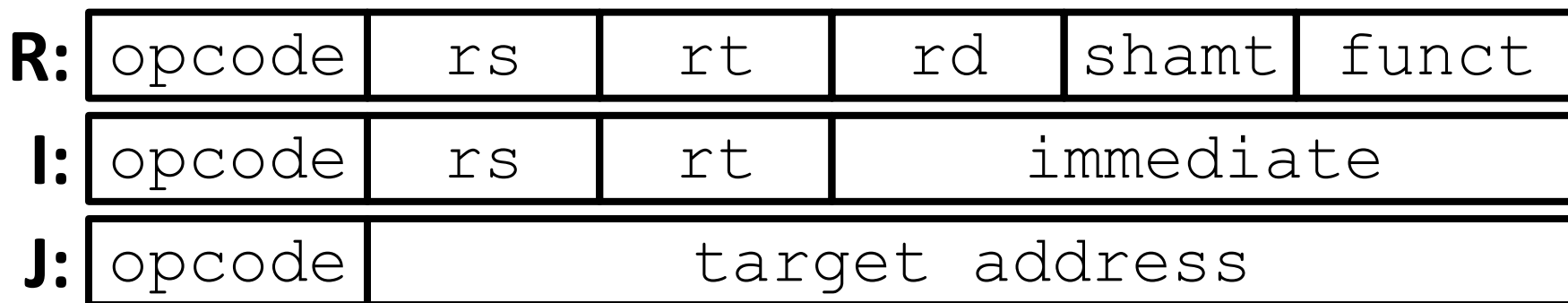


小节

现代计算机都是程序存储型的

- 指令与数据一样存储在主存中
- 读取指令与读取数据完全可以使用相同的硬件机制
- 指令与数据位于不同区域
 - 通过PC读取的“32位01串”都被CPU当做指令
 - 通过Load/Store指令读写的“32位01串”都被CPU当做数据

3类指令格式



B类指令使用PC相对寻址，J指令使用绝对地址寻址

作业

- 《计算机组成与设计》
- WORD
 - 2.26
- MARS
 - 2.23

提纲

- 内容主要取材：CS61C的7讲
 - <http://inst.eecs.berkeley.edu/~cs61c/su12>
- 存储型程序概念
- R-指令格式
- I-指令格式
 - 分支和PC相关寻址
- J-指令格式
- 汇编实战
- 反汇编实战

汇编

- ❑ 汇编是将汇编程序转换成二进制机器码的过程
- ❑ 每条指令的汇编基本步骤
 - ◆ S1: 标识出指令类型 (R/I/J)
 - ◆ S2: 标识出正确的域
 - ◆ S3: 用10进制表示各个域的值
 - ◆ S4: 把各个域的10进制转换为2进制
 - ◆ S5: 用16进制表示整个机器码



汇编实例

- 将下列汇编代码转换为相应的2进制机器码

地址 指令

800 Loop: sll \$t1,\$s3,2

804 addu \$t1,\$t1,\$s6

808 lw \$t0,0(\$t1)

812 beq \$t0,\$s5, Exit

816 addiu \$s3,\$s3,1

820 j Loop

Exit:

汇编实例

■ S1: 标识出指令类型

地址 指令

800 Loop: sll \$t1,\$s3,2

R

--	--	--	--	--	--

804 addu \$t1,\$t1,\$s6

R

--	--	--	--	--	--

808 lw \$t0,0(\$t1)

I

--	--	--	--

812 beq \$t0,\$s5,Exit

I

--	--	--	--

816 addiu \$s3,\$s3,1

I

--	--	--	--

820 j Loop

J

--	--

Exit:

汇编实例

■ S2: 标示出正确的域

地址 指令

800 Loop: sll \$t1,\$s3,2

R	opcode	rs	rt	rd	shamt	funct
---	--------	----	----	----	-------	-------

804 addu \$t1,\$t1,\$s6

R	opcode	rs	rt	rd	shamt	funct
---	--------	----	----	----	-------	-------

808 lw \$t0,0(\$t1)

I	opcode	rs	rt	immediate
---	--------	----	----	-----------

812 beq \$t0,\$s5,Exit

I	opcode	rs	rt	immediate
---	--------	----	----	-----------

816 addiu \$s3,\$s3,1

I	opcode	rs	rt	immediate
---	--------	----	----	-----------

820 j Loop

J	opcode	target address			
---	--------	----------------	--	--	--

Exit:

汇编实例

- S3: 用10进制表示各个域的值

地址 指令

800 Loop: sll \$t1, \$s3, 2

R	0	0	19	9	2	0
---	---	---	----	---	---	---

804 addu \$t1, \$t1, \$s6

R	0	9	22	9	0	33
---	---	---	----	---	---	----

808 lw \$t0, 0(\$t1)

I	35	9	8	0
---	----	---	---	---

812 beq \$t0, \$s5, Exit

I	4	8	21	2
---	---	---	----	---

816 addiu \$s3, \$s3, 1

I	8	19	19	1
---	---	----	----	---

820 j Loop

J	2	200
---	---	-----

Exit:

汇编实例

■ S4: 把10进制转换为2进制

地址 指令

800 Loop: sll \$t1,\$s3,2

R

000000	00000	10011	01001	00010	000000
--------	-------	-------	-------	-------	--------

804 addu \$t1,\$t1,\$s6

R

000000	01001	10110	01001	00000	100001
--------	-------	-------	-------	-------	--------

808 lw \$t0,0(\$t1)

I

100011	01001	01000	0000	0000	0000	0000
--------	-------	-------	------	------	------	------

812 beq \$t0,\$s5,Exit

I

000100	01000	10101	0000	0000	0000	0010
--------	-------	-------	------	------	------	------

816 addiu \$s3,\$s3,1

I

001000	00000	10011	0000	0000	0000	0001
--------	-------	-------	------	------	------	------

820 j Loop

J

000010	00	0000	0000	0000	0000	1100	1000
--------	----	------	------	------	------	------	------

Exit:

汇编实例

□ S5: 转成16进制

地址 指令

800 Loop: sll \$t1,\$s3,2

R 0x 0013 4880

804 addu \$t1,\$t1,\$s6

R 0x 0136 4821

808 lw \$t0,0(\$t1)

I 0x 8D28 0000

812 beq \$t0,\$s5, Exit

I 0x 1115 0002

816 addiu \$s3,\$s3,1

I 0x 2273 0001

820 j Loop

J 0x 0800 00C8

Exit:

提纲

- 内容主要取材：CS61C的7讲
 - <http://inst.eecs.berkeley.edu/~cs61c/su12>
- 存储型程序概念
- R-指令格式
- I-指令格式
 - 分支和PC相关寻址
- J-指令格式
- 汇编实战
- 反汇编实战

反汇编

- 反汇编是汇编的逆过程，即将指令二进制代码转换为汇编代码的过程
- 反汇编基本步骤
 - ◆ S1：用2进制表示指令
 - ◆ S2：根据opcode标示出指令类型（R/I/J）
 - ◆ S3：用10进制表示各个域的值
 - ◆ S4：用标识符表示各域，并添加相应的标号
 - ◆ S5：用汇编格式书写代码
 - ◆ S6：将汇编代码翻译为C
 - 通常很难翻译为可读性C代码！需要创造性！

反汇编实例

■ 把下来机器码翻译为汇编程序

地址	指令
0x00400000	0x00001025
0x00400004	0x0005402A
0x00400008	0x11000003
0x0040000C	0x00441020
0x00400010	0x20A5FFFF
0x00400014	0x08100001



反汇编实例

□ S1: 用2进制表示指令

地址	指令
0x00400000	00000000000000000000000010000000100101
0x00400004	00000000000000000010101000000000101010
0x00400008	00010001000000000000000000000000000011
0x0040000C	00000000001000100000010000001000000100000
0x00400010	00100000101001011111111111111111111111
0x00400014	000010000000100000000000000000000000001

反汇编实例

- S2: 根据opcode标示出指令类型

地址

指令

0x00400000	R	0000000	000000	000000	00010	000000	100101
0x00400004	R	0000000	000000	00101	01000	000000	101010
0x00400008	I	000100	01000	000000	0000000000000000011		
0x0040000C	R	0000000	00010	00100	00010	000000	100000
0x00400010	I	001000	00101	00101	111111111111111111		
0x00400014	J	000010	000001000000000000000000000001				

反汇编实例

- S3: 用10进制表示各个域的值

地址		指令						
0x00400000	R	0	0	0	2	0	37	
0x00400004	R	0	0	5	8	0	42	
0x00400008	I	4	8	0	+3			
0x0040000C	R	0	2	4	2	0	32	
0x00400010	I	8	5	5	-1			
0x00400014	J	2	0x0100001					

T
J指令的地址域保持为16进制

反汇编实例

□ S4: 用标识符表示各域

地址

指令

0x00400000	R	0	\$0	\$0	\$2	0	or
0x00400004	R	0	\$0	\$5	\$8	0	slt
0x00400008	I	beq	\$8	\$0	+3		
0x0040000C	R	0	\$2	\$4	\$2	0	add
0x00400010	I	addi	\$5	\$5	-1		
0x00400014	J	j	0x0100001				

反汇编实例

□ S5-1: 用汇编格式书写

地址	指令
0x00400000	or \$v0, \$0, \$0
0x00400004	slt \$t0, \$0, \$a1
0x00400008	beq \$t0, \$0, 3
0x0040000C	add \$v0, \$v0, \$a0
0x00400010	addi \$a1, \$a1, -1
0x00400014	j 0x01000001 # addr: 0x04000004



反汇编实例

- S5-1: 用汇编格式书写（用寄存器名有助于提高可读性）

地址	指令
0x00400000	or \$v0, \$0, \$0
0x00400004	slt \$t0, \$0, \$a1
0x00400008	beq \$t0, \$0, 3
0x0040000C	add \$v0, \$v0, \$a0
0x00400010	addi \$a1, \$a1, -1
0x00400014	j 0x01000001 # addr: 0x04000004

反汇编实例

■ S5-2: 用汇编格式书写（添加标号）

地址	指令
0x00400000	or \$v0, \$0, \$0
0x00400004	Loop: slt \$t0, \$0, \$a1
0x00400008	beq \$t0, \$0, 3
0x0040000C	add \$v0, \$v0, \$a0
0x00400010	addi \$a1, \$a1, -1
0x00400014	j Loop

反汇编实例

■ S5-2: 用汇编格式书写（添加标号）

```
        or      $v0, $0, $0      # initialize $v0 to 0
Loop:   slt     $t0, $0, $a1      # $t0 = 0 if 0 >= $a1
        beq     $t0, $0, 3        # exit if $a1 <= 0
        add     $v0, $v0, $a0     # $v0 += $a0
        addi    $a1, $a1, -1      # decrement $a1
        j       Loop
```

反汇编实例

□ S6: 将汇编程序转换为C程序

- ◆ 可以有很多种C代码对应到同一段汇编代码

```
/* a→$v0, b→$a0, c→$a1 */
```

```
a = 0;
```

```
while(c > 0) {
```

```
    a += b;
```

```
    c--;
```

```
}
```

□ 代码分析：循环c次，每次累加b，即b被累加c次

□ 代码功能：a = b x c